МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: АВЛ - ДЕРЕВЬЯ

Вариант № 18

Студент гр. 8304 Рыжиков А. В.
Преподаватель Фирсов К. В.

Санкт-Петербург
2019

1 Цель работы.

Получить опыт работы с АВЛ деревьями. Построить авл-дерево по исходным данным.

5-я лабораторная работа включает в себя задания нескольких типов, в зависимости от варианта.

В вариантах заданий 1-ой группы (кодирование и декодирование) на вход подаётся файл с закодированным или незакодированным содержимым. Требуется раскодировать или закодировать содержимое файла определённым алгоритмом.

В вариантах заданий 2-ой группы (БДП и хеш-таблицы) требуется:

1) По заданному файлу F (типа file of *Elem*), все элементы которого различны, построить структуру данных определённого типа – БДП или хеш-таблицу;
2) Выполнить одно из следующих действий:
   а) Для построенной структуры данных проверить, входит ли в неё элемент *e* типа *Elem*, и если не входит, то добавить элемент *e* в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.
   б) Для построенной структуры данных проверить, входит ли в неё элемент *e* типа *Elem*, и если входит, то удалить элемент *e* из структуры данных. Предусмотреть возможность повторного выполнения с другим элементом.
   в) Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран.
   г) Другое действие.

Варианту 18 соответсвует задание БДП: АВЛ-дерево; действие: 1+2в

2 Описание программы

АВЛ-дерево (AVL tree) –сбалансированное по высоте двоичное дерево поиска, в котором у любой вершины высота левого и правого поддеревьев различаются не более чем на 1.

Если вставка или удаление элемента приводит к нарушению сбалансированности дерева, то выполняется его балансировка. Коэффициент сбалансированности узла (balance factor) – это разность высот его левого и правого поддеревьев. В АВЛ-дереве коэффициент сбалансированности любого узла принимает значения из множества {-1, 0, 1}.

После добавления нового элемента необходимо обновить коэффициенты сбалансированности родительских узлов. Если любой родительский узел

принял значение -2 или 2, то необходимо выполнить балансировку поддерева путем поворота (rotation).

Типы поворотов:

1) Одиночный правый поворот (R-rotation, single right rotation)
2) Одиночный левый поворот (L-rotation, single left rotation)
3) Двойной лево-правый поворот (LR-rotation, double left-right rotation)
4) Двойной право-левый поворот (RL-rotation, double right-left rotation)

3.1 Описание структур данных.

```
struct Node {
    int key;
    unsigned char height;
    Node *left;
    Node *right;

    Node(int k) {
        key = k;
        left = right = 0;
        height = 1;
    }
};
```

Структура Node хранит указатель на левое и правое поддерево, хранит в себе ключ и высоту.

Функция insert.

Функция вставляет элемент в авл дерево в зависимости от его ключа и проводит балансировку дерева.

```
Node *insert(Node *p, int k) {
    if (!p) return new Node(k);
    if (k < p->key)
        p->left = insert(p->left, k);
    else
        p->right = insert(p->right, k);
    return balance(p);
}
```

Функция balance.

Функция приводит дерево в сбалансированное состояние. В зависимости от коэффициента балансировки элементов и нарушения сбалансированности применяются 4 выше описанные поворота.

```
const int right_overweight = 2;
const int left_overweight = -2;

Node *balance(Node *p) {
    fixHeight(p);
    if (bFactor(p) == right_overweight) {
        if (bFactor(p->right) < 0)
            p->right = rotateRight(p->right);
        return rotateLeft(p);
    }
    if (bFactor(p) == left_overweight) {
        if (bFactor(p->left) > 0)
            p->left = rotateLeft(p->left);
        return rotateRight(p);
    }
    return p;
}
```

Функция левого правого поворота

```
Node *rotateRight(Node *p) {
    Node *q = p->left;
    p->left = q->right;
    q->right = p;
    fixHeight(p);
    fixHeight(q);
    return q;
}

Node *rotateLeft(Node *q) {
    Node *p = q->right;
    q->right = p->left;
    p->left = q;
    fixHeight(q);
    fixHeight(p);
    return p;
}
```

Вспомогательные функции

Функция height возвращает глубину дерева.

Функция fixHeight устанавливает новое значение глубины.

Функция bFactor возвращает коэффициент сбалансированности.

```
unsigned char height(Node *p) {
    return p ? p->height : 0;
}

int bFactor(Node *p) {
    return height(p->right) - height(p->left);
}

void fixHeight(Node *p) {
    unsigned char hl = height(p->left);
    unsigned char hr = height(p->right);
    p->height = (hl > hr ? hl : hr) + 1;
}
```

Тесты.

В задании требуется всего лишь вывести в файл элементы авл дерева в порядке их возрастания. Для наглядности также будет выводится представление дерева в скобочной записи.

Тесты неудовлетворяющие условию ввода (любые неповторяющееся числа через запятую)

test #1 "x+x"

Unwanted characters

test #2 "ел1324"

Unwanted characters

test #3 "@dfd"

Unwanted characters

test #4 "1,2,3,4,a "

Unwanted characters и другие

Тесты удовлетворяющие условию ввода

test #5 "1,2,3"
(2L(1)R(3))
2
 1
 3
test #6 "1,2,3,4,5"
(2L(1)R(4L(3)R(5)))
2
 1
 4
  3
  5
test #7 "9,5,6,7,1"
(6L(5L(1))R(9L(7)))
6
 5
  1
 9

```
  7
test #8 "3,2,1"
(2L(1)R(3))
2
 1
 3
test #9 "9,5,6,8,2,3,4"
(6L(3L(2)R(5L(4)))R(9L(8)))
6
 3
  2
  5
   4
 9
  8
test #10 "1,2,3,4,5,6,7,8,9"
(4L(2L(1)R(3))R(6L(5)R(8L(7)R(9))))
4
 2
  1
  3
 6
  5
  8
   7
   9
test #11 "9,8,7,6,5,4,3,2,1"
(6L(4L(2L(1)R(3))R(5))R(8L(7)R(9)))
6
 4
  2
   1
   3
  5
 8
  7
  9
```

Вывод: *в ходе работы был получен опыт работы с авл- деревьями.*

Приложение

Код программы lab5.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

const int right_overweight = 2;
const int left_overweight = -2;
const int indent = 0;

struct Node {
    int key;
    unsigned char height;
    Node *left;
    Node *right;

    Node(int k) {
        key = k;
        left = right = 0;
        height = 1;
    }
};

unsigned char height(Node *p) {
    return p ? p->height : 0;
}

int bFactor(Node *p) {
    return height(p->right) - height(p->left);
}

void fixHeight(Node *p) {
    unsigned char hl = height(p->left);
    unsigned char hr = height(p->right);
    p->height = (hl > hr ? hl : hr) + 1;
}

Node *rotateRight(Node *p) {
    Node *q = p->left;
    p->left = q->right;
    q->right = p;
    fixHeight(p);
    fixHeight(q);
    return q;
}

Node *rotateLeft(Node *q) {
```

```
    Node *p = q->right;
    q->right = p->left;
    p->left = q;
    fixHeight(q);
    fixHeight(p);
    return p;
}

Node *balance(Node *p) {
    fixHeight(p);
    if (bFactor(p) == right_overweight) {
        if (bFactor(p->right) < 0)
            p->right = rotateRight(p->right);
        return rotateLeft(p);
    }
    if (bFactor(p) == left_overweight) {
        if (bFactor(p->left) > 0)
            p->left = rotateLeft(p->left);
        return rotateRight(p);
    }
    return p; // балансировка не нужна
}

Node *insert(Node *p, int k) {
    if (!p) return new Node(k);
    if (k < p->key)
        p->left = insert(p->left, k);
    else
        p->right = insert(p->right, k);
    return balance(p);
}

void printAVLTree(Node *p) {
    std::cout << "(";
    std::cout << p->key;
    if (p->left) {
        std::cout << "L";
        printAVLTree(p->left);
    }
    if (p->right) {
        std::cout << "R";
        printAVLTree(p->right);
    }
    std::cout << ")";
}

void printAVLTree2(Node *p, int maxHeight) {

    for (int i = 0; i < maxHeight; ++i) {
        std::cout << " ";
    }

    std::cout << p->key;
    std::cout << "\n";

    if (p->left) {
        printAVLTree2(p->left, maxHeight + 1);
    }
    if (p->right) {
        printAVLTree2(p->right, maxHeight + 1);
    }
```

```
}

void printAVLTree3(Node *p, std::ofstream &ofs) {

    if (p->left) {
        printAVLTree3(p->left, ofs);
    }
    ofs << p->key ;

    if (p->right) {
        printAVLTree3(p->right,  ofs);
    }

}

void deleteTree(Node *p){
    if (p->left) {
        deleteTree(p->left);
    }
    if (p->right) {
        deleteTree(p->right);
    }
    delete p;
}

bool checkExpression(std::string &name, int &my_length) {
    int length = name.length();
    for (int j = 0; j < length; ++j) {
        if (!isdigit(name[j]) && name[j] != ',') {
            return false;
        }
        if (name[j] == ',') {
            name[j] = ' ';
            (my_length)++;
        }
    }

    return true;
}


void mainCheck(std::string &name) {

    int my_lenght = 0;

    if (checkExpression(name, my_lenght)) {
        std::istringstream iss;
        std::string strvalues = name;
        iss.str(strvalues);

        int val2;
        iss >> val2;
        auto *myTree = new Node(val2);
        for (int i = 0; i < my_lenght; i++) {
            int val;
            iss >> val;
            myTree = insert(myTree, val);
        }
        printAVLTree(myTree);
        std::cout << '\n';
```

```
        myTree = balance(myTree);

        printAVLTree2(myTree, indent);

        std::ofstream ofs("C:\\Users\\Alex\\Desktop\\output.txt", std::ios_base::app);
        printAVLTree3(myTree,ofs);
        ofs << '\n';
        ofs.close();

        deleteTree(myTree);
    } else {
        std::cout << "Unwanted characters\n";
    }
}

int main() {

    int your_choose = 0;

    std::cout << "If you want to enter data from a file, enter \'1\'\n";
    std::cout << "If you want to enter data manually, enter \'2\'\n";

    std::cin >> your_choose;

    if (your_choose == 1) {
        std::ifstream fin;
        fin.open("C:\\Users\\Alex\\Desktop\\test5.txt");

        if (fin.is_open()) {
            std::cout << "Reading from file:" << "\n";

            int super_count = 0;

            while (!fin.eof()) {

                super_count++;

                std::string str;
                getline(fin, str);

                std::cout << "test #" << super_count << " \"" + str + "\"" << "\n";
                mainCheck(str);

            }
        } else {
            std::cout << "File not opened";
        }

        fin.close();
    } else {
        if (your_choose == 2) {
            std::cout << "Enter data \n";
            std::string str;
            std::cin >> str;
            mainCheck(str);
        }
    }

    return 0;
}
```